Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

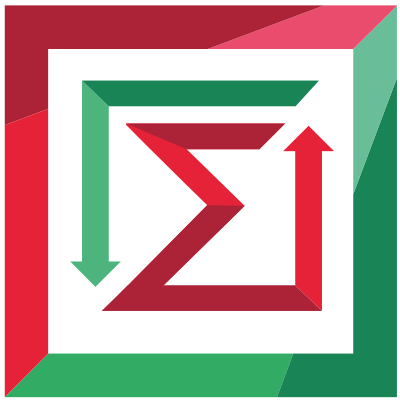
Новосибирский государственный
технический университет
НЭТИ

Кафедра теоретической и прикладной информатики

Лабораторная работа № 1

по дисциплине «Методы оптимизации»

МЕТОДЫ ОДНОМЕРНОГО ПОИСКА



Факультет: ПМИ

Группа: ПМИ-81

Бригада: 5

Студенты: Кайда Даниил

Парышков Дмитрий

Преподаватели: Постовалов Сергей Николаевич

Лемешко Борис Юрьевич

Новосибирск
2021

1. Цель работы

Ознакомиться с методами одномерного поиска - метод дихотомии, метод золотого сечения, метод Фибоначчи - используемыми в многомерных методах минимизации функций n переменных. Сравнить различные алгоритмы по эффективности на тестовых примерах.

2. Задание

Реализовать методы дихотомии, золотого сечения, Фибоначчи, исследовать их сходимость и провести сравнение по числу вычислений функции для достижения заданной точности ε от 1E-1 до 1E-7. Построить график зависимости количества вычислений минимизируемой функции от десятичного логарифма задаваемой точности ε. Реализовать алгоритм поиска интервала, содержащего минимум функции.

$$f(x) = (x-5)^2$$

3. Ход работы

Реализовали методы дихотомии, золотого сечения, Фибоначчи и алгоритм поиска интервала, содержащего минимум функции.

Общий класс для всех методов

```
class Method
{
    protected DataTable DataTable;

    protected Data Data;

    protected Function Func;

    protected int NumberOfIterationsObjectiveFunction;

    public int getNumberOfIterationsObjectiveFunction()
    {
        return NumberOfIterationsObjectiveFunction;
    }

    public void ShowTable(double Eps)
    {
        DataTable.DrawTable(Eps);
    }
}
```

Метод дихометрии

```
class DichometricsMethod : Method
    {
        public DichometricsMethod()
        {
            DataTable = new DataTable();
            Func = new Function();
            Data = new Data(0, 0, 0, 0, 0, 1);
        }

        public void Do(double a, double b, double Eps = 0.001)
        {
            DataTable.ClearTable();
            double Delta = Eps / 10;
            Data.a = a;
            Data.b = b;
            Data.x1 = (Data.a + Data.b - Delta) / 2;
            Data.x2 = (Data.a + Data.b + Delta) / 2;
            Data.fx1 = Func.Value(Data.x1);
            Data.fx2 = Func.Value(Data.x2);
            DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a, Data.b);

            NumberOfIterationsObjectiveFunction = 0;
            while
(DataTable.Table[NumberOfIterationsObjectiveFunction++].difference_ab > Eps)
            {
                if (Data.fx1 < Data.fx2)
                    Data.b = Data.x2;
                else
                    Data.a = Data.x1;
                Data.x1 = (Data.a + Data.b - Delta) / 2;
                Data.x2 = (Data.a + Data.b + Delta) / 2;
                Data.fx1 = Func.Value(Data.x1);
                Data.fx2 = Func.Value(Data.x2);
                DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a, Data.b);
            }
            NumberOfIterationsObjectiveFunction *= 2;
        }
     }
```

Метод золотого сечения

```
   class GoldenSectionMethod : Method
       {
           public GoldenSectionMethod()
           {
               DataTable = new DataTable();
               Func = new Function();
               Data = new Data(0, 0, 0, 0, 0, 1);
           }

           public void Do(double a, double b, double Eps = 0.001)
           {
               DataTable.ClearTable();
               Data.a = a;
               Data.b = b;
               Data.difference_ab = Math.Abs(Data.b - Data.a);
               Data.x1 = Data.a + 0.381966011 * Data.difference_ab;
               Data.x2 = Data.a + (1 - 0.381966011) * Data.difference_ab;
               Data.fx1 = Func.Value(Data.x1);
```

```
            Data.fx2 = Func.Value(Data.x2);
            DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a, Data.b);

            NumberOfIterationsObjectiveFunction = 0;
            while
(DataTable.Table[NumberOfIterationsObjectiveFunction++].difference_ab > Eps)
            {
                if (Data.fx1 < Data.fx2)
                {
                    Data.b = Data.x2;
                    Data.x2 = Data.x1;
                    Data.fx2 = Data.fx1;
                    Data.difference_ab = Math.Abs(Data.b - Data.a);
                    Data.x1 = Data.a + 0.381966011 * Data.difference_ab;
                    Data.fx1 = Func.Value(Data.x1);
                }
                else
                {
                    Data.a = Data.x1;
                    Data.x1 = Data.x2;
                    Data.fx1 = Data.fx2;
                    Data.difference_ab = Math.Abs(Data.b - Data.a);
                    Data.x2 = Data.a + (1 - 0.381966011) * Data.difference_ab;
                    Data.fx2 = Func.Value(Data.x2);
                }

                DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a,
Data.b);
            }
        }
           }
```

Метод Фибоначчи

```
class FibonacciMethod : Method
    {
        List<int> F;
        public FibonacciMethod()
        {
            DataTable = new DataTable();
            Func = new Function();
            Data = new Data(0, 0, 0, 0, 0, 1);
            F = new List<int>();
            F.Add(1);
            F.Add(1);

            const int n = 150;
            for (int i = 0; i < n; i++)
                F.Add(F[i] + F[i + 1]);
        }

        int Definition_n(double value)
        {
            int n = 0;

            while (value > F[n++ + 2]) ;

            return n;
        }

        public void Do(double a, double b, double Eps = 0.001)
        {
            DataTable.ClearTable();
            Data.a = a;
```

```
            Data.b = b;
            Data.difference_ab = Math.Abs(Data.b - Data.a);

            int n = Definition_n(Data.difference_ab / Eps);
            int k = 0;
            double temp = F[n - k];
            double temp_2 = F[n + 2];
            double temp_3 = temp / temp_2;
            Data.x1 = Data.a + temp_3 * Data.difference_ab;
            temp = F[++n - k + 1];
            temp_3 = temp / temp_2;
            Data.x2 = Data.a + temp_3 * Data.difference_ab;

            Data.fx1 = Func.Value(Data.x1);
            Data.fx2 = Func.Value(Data.x2);
            DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a, Data.b);

            NumberOfIterationsObjectiveFunction = 0;
            while
(DataTable.Table[NumberOfIterationsObjectiveFunction++].difference_ab > Eps)
            {
                k++;
                if (Data.fx1 < Data.fx2)
                {
                    Data.b = Data.x2;
                    Data.x2 = Data.x1;
                    Data.fx2 = Data.fx1;
                    temp = F[n - k ];
                    temp_2 = F[n + 2];
                    temp_3 = temp / temp_2;
                    Data.x1 = Data.a + temp_3 * Data.difference_ab;
                    Data.fx1 = Func.Value(Data.x1);
                }
                else
                {
                    Data.a = Data.x1;
                    Data.x1 = Data.x2;
                    Data.fx1 = Data.fx2;
                    temp = F[n - k + 1];
                    temp_3 = temp / temp_2;
                    Data.x2 = Data.a + temp_3 * Data.difference_ab;
                    Data.fx2 = Func.Value(Data.x2);
                }
                DataTable.Add(Data.x1, Data.x2, Data.fx1, Data.fx2, Data.a,
Data.b);
            }

        }
           }
```

Алгоритм поиска интервала, содержащего минимум функции

```
    class IntervalSearch
    {
        List<double> x;

        List<double> fx;

        Function Func;

        double a, b;

        public IntervalSearch()
        {
```

```
            Func = new Function();

            x = new List<double>();

            fx = new List<double>();
        }

        public void Search(double Xzero, double Delta)
        {
            x.Clear();
            fx.Clear();

            double h = Delta;

            int k = 0;

            x.Add(Xzero);
            fx.Add(Func.Value(x[k]));

            if (fx[0] > Func.Value(x[k] + Delta))
            {
                x.Add(x[k] + Delta);
                h = Delta;
            }
            else
                if ((fx[0] > Func.Value(x[k] - Delta)))
            {
                x.Add(x[k] - Delta);
                h = -Delta;
            }
            fx.Add(Func.Value(x[k + 1]));
            do
            {
                k++;
                h *= 2;
                x.Add(x[k] + h);
                fx.Add(Func.Value(x[k + 1]));

            } while (fx[k] > fx[k + 1]);

            a = x[k - 1];
            b = x[k + 1];
        }

        public void ShowResult()
        {
            Console.WriteLine("{0,3}    {1,12}     {2,12}", "i", "x", "f(x)");
            for (int i = 0; i < x.Count; i++)
                Console.WriteLine("{0,3}   {1: 0.0000000000}   {2: 0.0000000000}",
i, x[i], fx[i]);
            Console.WriteLine("The interval containing the minimum of the
function: [{0:0.00000},{1:0.00000}]", x[x.Count - 3], x[x.Count - 1]);
        }
                }
```

4. Результаты исследований

Метод дихометрии

```
Dichometrics Method:
Number Of Iterations: 58
Calculation accuracy - 1E-07
  i           x1           x2                    fx1                    fx2             ai             bi          bi - ai    (b-a)_i-1 / (b-a)_i
  0    9,00000000   9,00000001   63,99999920000000000   64,00000080000000000      -2,00000000     20,00000000     22,00000000      1,00000000
  1    3,50000000   3,50000001   6,24999998750000000   6,25000037500000000      -2,00000000      9,00000001     11,00000001      0,00000000
  2    0,75000000   0,75000001   0,06250000624999900   0,06249999625000000      -2,00000000      3,50000001      5,50000001      2,00000000
  3    2,12500000   2,12500001   1,26562499578125000   1,26562501828125000      0,75000000      3,50000001      2,75000001      2,00000000
  4    1,43750000   1,43750001   0,19140624863281300   0,19140625738281300      0,75000000      2,12500001      1,37500001      1,99999999
  5    1,09375000   1,09375001   0,00878906223632817 0   0,00878906411132823 0   0,75000000      1,43750001      0,68750001      1,99999999
  6    0,92187500   0,92187501   0,00610351583251951 0   0,00610351427001959 0   0,75000000      1,09375001      0,34375001      1,99999997
  7    1,00781250   1,00781251   0,00006103513488770 1   0,00006103529113777 3   0,92187500      1,09375001      0,17187501      1,99999994
  8    0,96484375   0,96484376   0,00123596200881956 0   0,00123596130569464 0   0,92187500      1,00781251      0,08593751      1,99999988
  9    0,98632812   0,98632813   0,00018692020313262 4   0,00018691992969519 9   0,96484375      1,00781251      0,04296876      1,99999977
 10    0,99707031   0,99707032   0,00000858307682991 1   0,00000858301823623 4   0,98632812      1,00781251      0,02148438      1,99999953
 11    1,00244140   1,00244141   0,00000596045781374 3   0,00000596050664194 0   0,99707031      1,00781251      0,01074220      1,99999907
 12    0,99975586   0,99975587   0,00000005960531056 1   0,00000005960042782 1   0,99707031      1,00244141      0,00537110      1,99999814
 13    1,00109863   1,00109864   0,00000120699105933 6   0,00000120701303206 4   0,99975586      1,00244141      0,00268556      1,99999628
 14    1,00042724   1,00042725   0,00000018253805924 4   0,00000018254660423 9   0,99975586      1,00109864      0,00134278      1,99999255
 15    1,00009155   1,00009156   0,00000000838165347 7   0,00000000838348460 4   0,99975586      1,00042725      0,00067140      1,99998511
 16    0,99992370   0,99992371   0,00000000582097416 2   0,00000000581944835 6   0,99975586      1,00009156      0,00033570      1,99997021
 17    1,00000763   1,00000764   0,00000000005818685 5   0,00000000005833951 6   0,99992370      1,00009156      0,00016786      1,99994043
 18    0,99996567   0,99996568   0,00000000117879876 8   0,00000000117811219 5   0,99992370      1,00000764      0,00008393      1,99988086
 19    0,99998665   0,99998666   0,00000000017829737 6   0,00000000017803042 0   0,99996567      1,00000764      0,00004197      1,99976174
 20    0,99999714   0,99999715   0,00000000000819325 7   0,00000000000813610 9   0,99998665      1,00000764      0,00002099      1,99952360
 21    1,00000238   1,00000239   0,00000000000567784 1   0,00000000000572559 8   0,99999714      1,00000764      0,00001050      1,99904766
 22    0,99999976   0,99999977   0,00000000000005749 6   0,00000000000005280 0   0,99999714      1,00000239      0,00000526      1,99809713
 23    1,00000107   1,00000108   0,00000000000114815 5   0,00000000000116968 5   0,99999976      1,00000239      0,00000263      1,99620148
 24    1,00000042   1,00000043   0,00000000000017294 7   0,00000000000018136 4   0,99999976      1,00000108      0,00000132      1,99243171
 25    1,00000009   1,00000010   0,00000000000000775 2   0,00000000000000961 2   0,99999976      1,00000043      0,00000067      1,98497711
 26    0,99999992   0,99999993   0,00000000000000575 6   0,00000000000000433 9   0,99999976      1,00000010      0,00000034      1,97039892
 27    1,00000001   1,00000002   0,00000000000000003 7   0,00000000000000025 9   0,99999992      1,00000010      0,00000017      1,94249991
 28    0,99999997   0,99999998   0,00000000000000121 7   0,00000000000000062 0   0,99999992      1,00000002      0,00000009      1,89125280
Result: x = 0,999999970108751
```

Метод золотого сечения

```
GoldenSection Method:
Number Of Iterations: 41
Calculation accuracy - 1E-07
```

i	x1	x2	fx1	fx2	ai	bi	bi - ai	(b-a)_i-1 / (b-a)_i
0	6,40325224	11,59674776	29,195134790678000000	112,291063046678000000	-2,00000000	20,00000000	22,00000000	1,00000000
1	3,19349550	6,40325224	4,811422524736560000	29,195134790678000000	-2,00000000	11,59674776	13,59674776	0,00000000
2	1,20975674	3,19349550	0,043997889263742600	4,811422524736560000	-2,00000000	6,40325224	8,40325224	1,61803399
3	- 0,01626124	1,20975674	1,032786906517050000	0,043997889263742600	-2,00000000	3,19349550	5,19349550	1,61803399
4	1,20975674	1,96747752	0,043997889263742600	0,936012760008655000	-0,01626124	3,19349550	3,20975674	1,61803399
5	0,74145954	1,20975674	0,066843167858761000	0,043997889263742600	-0,01626124	1,96747752	1,98373876	1,61803399
6	1,20975674	1,49918033	0,043997889263742600	0,249180998462346000	0,74145954	1,96747752	1,22601798	1,61803399
7	1,03088313	1,20975674	0,000953767608582202	0,043997889263742600	0,74145954	1,49918033	0,75772078	1,61803399
8	0,92033315	1,03088313	0,006346806238656900	0,000953767608582202	0,74145954	1,20975674	0,46829720	1,61803400
9	1,03088313	1,09920677	0,000953767608582202	0,009841982536949990	0,92033315	1,20975674	0,28942358	1,61803399
10	0,98865679	1,03088313	0,000128668306213842	0,000953767608582202	0,92033315	1,09920677	0,17887361	1,61803399
11	0,96255949	0,98865679	0,001401792005722940	0,000128668306213842	0,92033315	1,03088313	0,11054997	1,61803396
12	0,98865679	1,00478582	0,000128668306213842	0,000022904068920225	0,96255949	1,03088313	0,06832364	1,61803399
13	1,00478582	1,01475410	0,000022904068920225	0,000217683586316709	0,98865679	1,03088313	0,04222633	1,61803395
14	0,99862508	1,00478582	0,000001890405398451	0,000022904068920225	0,98865679	1,01475410	0,02609731	1,61803399
15	0,99481753	0,99862508	0,000026857953875804	0,000001890405398451	0,98865679	1,00478582	0,01612902	1,61803392
16	0,99862508	1,00097827	0,000001890405398451	0,000000957018631284	0,99481753	1,00478582	0,00996829	1,61803399
17	1,00097827	1,00243263	0,000000957018631284	0,000005917671176829	0,99862508	1,00478582	0,00616074	1,61803388
18	1,00007943	1,00097827	0,000000006309636442	0,000000957018631284	0,99862508	1,00243263	0,00380755	1,61803399
19	0,99952392	1,00007943	0,000000226652388974	0,000000006309636442	0,99862508	1,00097827	0,00235319	1,61803381
20	1,00007943	1,00042276	0,000000006309636442	0,000000178725743969	0,99952392	1,00097827	0,00145435	1,61803399
21	0,99986725	1,00007943	0,000000017623608255	0,000000006309636442	0,99952392	1,00042276	0,00089884	1,61803399
22	1,00007943	1,00021057	0,000000006309636442	0,000000044340717426	0,99986725	1,00042276	0,00055551	1,61803399
23	0,99999839	1,00007943	0,000000000002608106	0,000000006309636442	0,99986725	1,00021057	0,00034333	1,61803399
24	0,99994829	0,99999839	0,000000002673474876	0,000000000002608106	0,99986725	1,00007943	0,00021219	1,61803522
25	0,99999839	1,00002934	0,000000000002608106	0,000000000860989736	0,99994829	1,00007943	0,00013114	1,61803399
26	0,99997925	0,99999839	0,000000000430478130	0,000000000002608106	0,99994829	1,00002934	0,00008105	1,61803399
27	0,99999839	1,00001021	0,000000000002608106	0,000000000104238432	0,99997925	1,00002934	0,00005009	1,61803399
28	0,99999108	0,99999839	0,000000000079623173	0,000000000002608106	0,99997925	1,00001021	0,00003096	1,61803399
29	0,99999839	1,00000290	0,000000000002608106	0,000000000008419302	0,99999108	1,00001021	0,00001913	1,61803399
30	0,99999559	0,99999839	0,000000000019417381	0,000000000002608106	0,99999108	1,00000290	0,00001182	1,61803399
31	0,99999839	1,00000011	0,000000000002608106	0,000000000000012133	0,99999559	1,00000290	0,00000731	1,61803399
32	1,00000011	1,00000118	0,000000000000012133	0,000000000001383984	0,99999839	1,00000290	0,00000452	1,61806963
33	0,99999945	1,00000011	0,000000000000301123	0,000000000000012133	0,99999839	1,00000118	0,00000279	1,61803399
34	1,00000011	1,00000052	0,000000000000012133	0,000000000000267775	0,99999945	1,00000118	0,00000173	1,61803399
35	0,99999986	1,00000011	0,000000000000020019	0,000000000000012133	0,99999945	1,00000052	0,00000107	1,61803399
36	1,00000011	1,00000027	0,000000000000012133	0,000000000000070634	0,99999986	1,00000052	0,00000066	1,61803399
37	1,00000001	1,00000011	0,000000000000000198	0,000000000000012133	0,99999986	1,00000027	0,00000041	1,61803399
38	0,99999995	1,00000001	0,000000000000002059	0,000000000000000198	0,99999986	1,00000011	0,00000025	1,61842934
39	1,00000001	1,00000005	0,000000000000000198	0,000000000000002575	0,99999995	1,00000011	0,00000016	1,61803399
40	0,99999999	1,00000001	0,000000000000000075	0,000000000000000198	0,99999995	1,00000005	0,00000009	1,61803399

```
Result: x = 1,0000000026879052
```

Метод Фибоначчи

```
Fibonacci Method:
Number Of Iterations: 41
Calculation accuracy - 1E-07
```

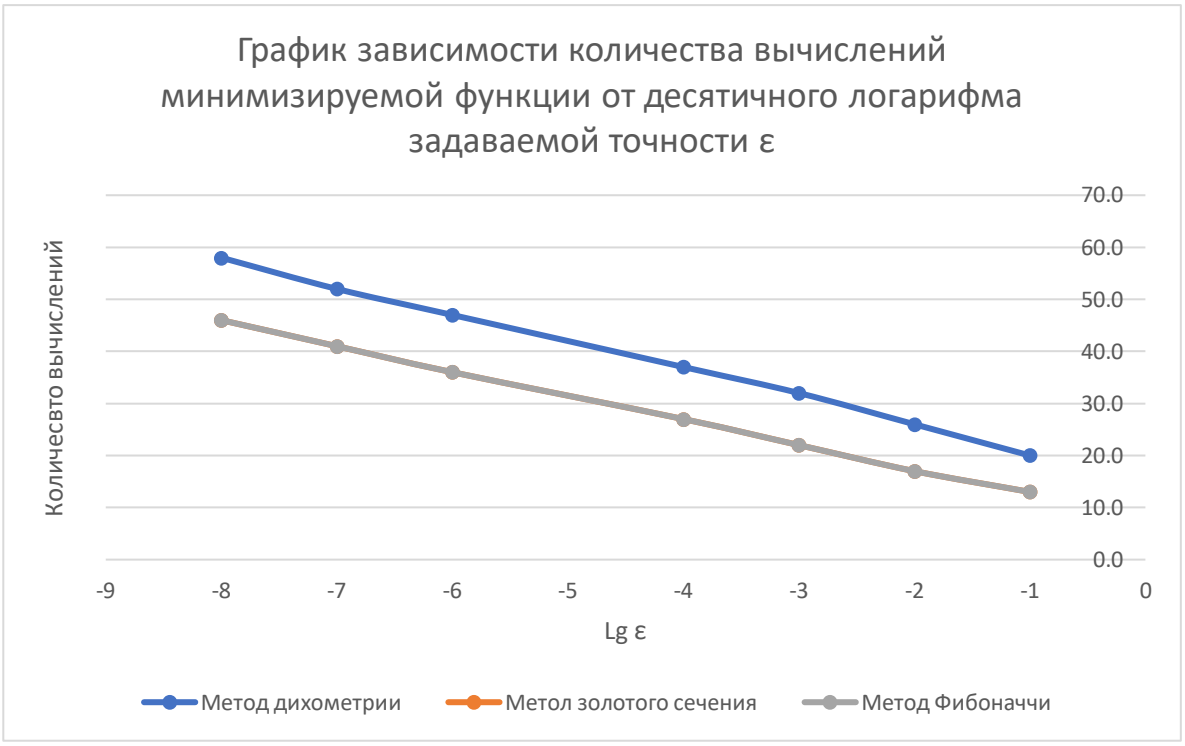
i	x1	x2	fx1	fx2	ai	bi	bi - ai	(b-a)_i-1 / (b-a)_i
0	6,40325225	20,00000000	29,195134850138800000	361,000000000000000000	-2,00000000	20,00000000	22,00000000	1,00000000
1	3,19349550	6,40325225	4,811422530434910000	29,195134850138800000	-2,00000000	20,00000000	22,00000000	0,00000000
2	1,20975674	3,19349550	0,043997891027122700	4,811422530434910000	-2,00000000	6,40325225	8,40325225	2,61803399
3	- 0,01626124	1,20975674	1,032786902868540000	0,043997891027122700	-2,00000000	3,19349550	5,19349550	1,61803399
4	1,20975674	1,96747752	0,043997891027122700	0,936012761335364000	-0,01626124	3,19349550	3,20975674	1,61803399
5	0,74145954	1,20975674	0,066843166893138100	0,043997891027122700	-0,01626124	1,96747752	1,98373876	1,61803399
6	1,20975674	1,49918033	0,043997891027122700	0,249180999291412000	0,74145954	1,96747752	1,22601798	1,61803399
7	1,03088313	1,20975674	0,000953767711166878	0,043997891027122700	0,74145954	1,49918033	0,75772078	1,61803399
8	0,92033316	1,03088313	0,006346805780281980	0,000953767711166878	0,74145954	1,20975674	0,46829720	1,61803399
9	1,03088313	1,09920677	0,000953767711166878	0,009841983256060350	0,92033316	1,20975674	0,28942358	1,61803399
10	0,98865680	1,03088313	0,001128668233456962	0,000953767711166878	0,92033316	1,09920677	0,17887361	1,61803399
11	0,96255949	0,98865680	0,001401791823011950	0,001128668233456962	0,92033316	1,03088313	0,11054997	1,61803399
12	0,98865680	1,00478582	0,001128668233456962	0,000222904087502473	0,96255949	1,03088313	0,06832364	1,61803399
13	1,00478582	1,01475411	0,000222904087502473	0,000217683652441177	0,98865680	1,03088313	0,04222633	1,61803399
14	0,99862508	1,00478582	0,000001890397576457	0,000222904087502473	0,98865680	1,01475411	0,02609731	1,61803399
15	0,99481754	0,99862508	0,000026857925603569	0,000001890397576457	0,98865680	1,00478582	0,01612902	1,61803399
16	0,99862508	1,00097828	0,000001890397576457	0,000000957023012535	0,99481754	1,00478582	0,00996828	1,61803399
17	1,00097828	1,00243263	0,000000957023012535	0,000005917682292690	0,99862508	1,00478582	0,00616074	1,61803399
18	1,00007944	1,00097828	0,000000006310054514	0,000000957023012535	0,99862508	1,00243263	0,00380755	1,61803399
19	0,99952392	1,00007944	0,000000226649899920	0,000000006310054514	0,99862508	1,00097828	0,00235319	1,61803399
20	1,00007944	1,00042276	0,000000006310054514	0,000000178727758333	0,99952392	1,00097828	0,00145435	1,61803399
21	0,99986725	1,00007944	0,000000017622937503	0,000000006310054514	0,99952392	1,00042276	0,00089884	1,61803399
22	1,00007944	1,00021057	0,000000006310054514	0,000000044341744192	0,99986725	1,00042276	0,00055551	1,61803399
23	0,99999839	1,00007944	0,000000000002600057	0,000000006310054514	0,99986725	1,00021057	0,00034333	1,61803399
24	0,99994830	0,99999839	0,000000002673209685	0,000000000002600057	0,99986725	1,00007944	0,00021219	1,61803400
25	0,99999839	1,00002935	0,000000000002600057	0,000000000861142474	0,99994830	1,00007944	0,00013114	1,61803396
26	0,99997925	0,99999839	0,000000000430371341	0,000000000002600057	0,99994830	1,00002935	0,00008105	1,61803406
27	0,99999839	1,00001021	0,000000000002600057	0,000000000104291176	0,99997925	1,00002935	0,00005009	1,61803381
28	0,99999108	0,99999839	0,000000000079577442	0,000000000002600057	0,99997925	1,00001021	0,00003096	1,61803445
29	0,99999839	1,00000290	0,000000000002600057	0,000000000008434119	0,99999108	1,00001021	0,00001913	1,61803279
30	0,99999560	0,99999839	0,000000000019395165	0,000000000002600057	0,99999108	1,00000290	0,00001182	1,61803714
31	0,99999839	1,00000011	0,000000000002600057	0,000000000000012686	0,99999560	1,00000290	0,00000731	1,61802575
32	1,00000011	1,00000118	0,000000000000012686	0,000000000001390172	0,99999839	1,00000290	0,00000452	1,61805556
33	0,99999945	1,00000011	0,000000000000298164	0,000000000000012686	0,99999839	1,00000118	0,00000279	1,61797753
34	1,00000011	1,00000052	0,000000000000012686	0,000000000000270797	0,99999945	1,00000118	0,00000173	1,61818182
35	0,99999986	1,00000011	0,000000000000019125	0,000000000000012686	0,99999945	1,00000052	0,00000107	1,61764706
36	1,00000011	1,00000027	0,000000000000012686	0,000000000000072607	0,99999986	1,00000052	0,00000066	1,61904762
37	1,00000002	1,00000011	0,000000000000000344	0,000000000000012686	0,99999986	1,00000027	0,00000041	1,61538462
38	0,99999996	1,00000002	0,000000000000001953	0,000000000000000344	0,99999986	1,00000011	0,00000025	1,62500000
39	1,00000002	1,00000005	0,000000000000000344	0,000000000000002490	0,99999996	1,00000011	0,00000016	1,60000000
40	0,99999999	1,00000002	0,000000000000000165	0,000000000000000344	0,99999996	1,00000005	0,00000009	1,66666666

```
Result: x = 1,0000000028514047
```

Количество вычислений целевой функции для каждого метода в зависимости от заданной точности

Number Of Iterations Objective Function:

Name Method | Eps	0,1	0,01	0,001	0,0001	1E-05	1E-06	1E-07
Dichometrics	18	26	32	38	46	52	58
Golden Section	13	17	22	27	32	37	41
Fibonacci	13	17	22	27	32	37	41



Вывод

График показывает линейную зависимость между количеством вычислений минимизируемой функции от логарифма задаваемой точности ε, следовательно, чем точнее необходимо решение, тем большее количество итераций необходимо сделать.

Количество вычислений минимизируемой функции для равной точности у методов золотого сечения и Фибоначчи меньше, чем у метода дихотомии, но количество итераций у метода дихотомии меньше. Методы золотого сечения и Фибоначчи эффективно использовать для функций, вычисление которых затратно по времени или/и ресурсам.

Метод золотого сечения и метод Фибоначчи обладают приблизительно одинаковой скоростью сходимости, лучшей, чем у метода дихотомии. Метод Фибоначчи также позволяет заранее предсказать количество итераций ценой большей вычислительной сложности, но при этом необходимо вычислять значение функции Фибоначчи, что несёт дополнительные затраты по вычислительной мощности.

Таблица, показывающая процесс поиска интервала, содержащего минимум:

$$\delta = 0.1, x_0 = 4$$

i	x_i	$f(x_i)$
0	0	25.00
1	0.1	24.01
2	0.3	22.09
3	0.7	18.49
4	1.5	12.25
5	3.1	3.61
6	6.3	1.69
7	12.7	59.29

Интервал, содержащий минимум: $[\mathbf{3.1}, \mathbf{12.7}]$

Вывод

При приближении начального приближения к точке минимума количество итераций уменьшается. Как только точка минимума была пройдена, итерационный процесс заканчивается.